AMATH 582 HW5 Seperation of Videos using DMD

Huasong Zhang
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
huasongz@uw.edu

Abstract

In this assignment, our task is to use DMD method to reduce the high dimensional data matrix into low-dimensional. By using this method, we can separate the video into background (low rank) and foreground (sparse) components in time, so that we can observe and focus on the the movement of the object more clearly.

1 Introduction and Overview

In our daily life, there are many complex things that are represented in high dimensional data, which are not easy to track and observe. In order to focus more on the things that we really care about, people have found a new way to transfer high dimensional data into low dimensional while the thing can also be well presented to deal with problem like that. This method is called dynamical mode decomposition (DMD). This method is a combination of dimension reduction in space and Fourier transform in time, so that a spatio-temporal feature of the movement can be extracted and easily interpreted. Using DMD, we can even predict what is going on in the future. Therefore, DMD is a pretty powerful method in data-driven analysis.

2 Theoretical Background

For DMD, we consider the data collected are from a dynamical system, which has the form

$$\frac{dx}{dt} = f(x, t, \mu) \tag{1}$$

x is a n dimensional vector and we collect the data every Δt in time. Therefore, we have

$$x_{k+1} = F(x_k) \tag{2}$$

where F is some function that takes the current x to the next. Since the actual movement is hard to capture, therefore, we have

$$y_k = g(x_k) \tag{3}$$

to be the measure of the movement. In order to solve this dynamical system, we approximately consider it as linear, which is

$$\frac{dx}{dt} = Ax \tag{4}$$

where the x has the well known solution of

$$x(t) = \sum_{k=1}^{n} \phi_k e^{\omega_k t} b_k = \Phi e^{\Omega t} b \tag{5}$$

where ϕ_k and ω_k are eigenvectors and eigenvalues of A, and b_k are the the coordinates of x(0) in the eigenvector basis. Based on the linear dynamical system above, we have

$$x_{k+1} = \mathbf{A} x_k \tag{6}$$

where

$$\mathbf{A} = e^{A\Delta t} \tag{7}$$

A is the continuous-time dynamics in Equation 4 and **A** is the discrete-time map in Equation 6. Therefore, the solution can be written as:

$$x_k = \sum_{i=1}^{r} \phi_i \lambda_i^k b_j = \Phi \Lambda^k b \tag{8}$$

where ϕ and λ are eigenvectors and eigenvalues of **A**, and b are still the coefficients of the initial condition in the eigenvector basis. Therefore we have initial condition

$$x_1 = \Phi b \tag{9}$$

since when time equals to 0, $e^0 = 1$, the eigenvalue term disappears. The DMD algorithm is trying to find the **A** which makes

$$||x_{k+1} - \mathbf{A}x_k||_2 \tag{10}$$

is minimized. Now, we have transformed a high-dimensional continuous dynamical system to a low-dimensional discrete dynamical system which is easier to solve. Now, we need to use the data that we collected. Suppose the data matrix X is n by m, where n is the dimension of each data vector, and m is the number of data. We take the first one to the second end data to be X and second one to the end of the data to be X', and they are:

$$X = \begin{bmatrix} | & | & ... & | \\ x_1 & x_2 & ... & x_{m-1} \\ | & | & ... & | \end{bmatrix} X' = \begin{bmatrix} | & | & ... & | \\ x_2 & x_3 & ... & x_m \\ | & | & ... & | \end{bmatrix} \tag{11}$$

X' represents what is going on after X, which is time Δt later. So we have

$$X' \approx \mathbf{A}X \tag{12}$$

Therefore, we can solve for A, which is

$$\mathbf{A} = X'X^\dagger \tag{13}$$

where $X^\dagger$is the pseudo-inverse of X. After we take SVD on X, we have

$$X = U\Sigma V^* \tag{14}$$

Plugging in to the **A** equation we have

$$\mathbf{A} = X'V\Sigma^{-1}U^* \tag{15}$$

Instead of computing the full rank A, it is more convenient to work on the A with r dimension only. So the equation above becomes:

$$\tilde{\mathbf{A}} = U^*\mathbf{A}U = U^*X'V\Sigma^{-1}U^*U = U^*X'V\Sigma^{-1} \tag{16}$$

Based on the derivation we have done before, now we have:

$$\tilde{x_{k+1}} = \tilde{\mathbf{A}}\tilde{x_k} \tag{17}$$

Then we do eigenvalue decomposition of $\tilde{\mathbf{A}}$, we get:

$$\tilde{\mathbf{A}}\mathbf{W} = \mathbf{W}\Lambda \tag{18}$$

where **W** is the eigenvectors matrix of $\tilde{\mathbf{A}}$ and Λ is the corresponding eigenvalue matrix. Last, we can reconstruct eigendecomposition of **A** by using Λ and **W**. Eigenvalues of A are given by Λ and eigenvectors of **A** are reconstructed using the following fomula:

$$\Phi = X'V\Sigma^{-1}\mathbf{W} \tag{19}$$

We just change **A** to Φ and U^* to **W** in the Equation 15. Now, we can approximate x(t) using the low rank structure:

$$x(t) \approx \sum_{i=1}^{r} \phi_k e^{\omega_k t} b_k = \Phi e^{\Omega t} b \tag{20}$$

where b is calculated using the Equation 9. This equation is the same as Equation 5. The only difference is the matrices used here only have dimension r instead of n. This is the process of how we get the solution by using low-dimensional data.

3 Algorithm Implementation and Development

To implement the DMD algorithm, the first step is to load the data using VideoReader. Then the nFrames will displace how time is discretized; it is the number of snap shots in total in the entire video. Then in the for loop, I go through each image, change it from rgb to grayscale, and then reshape each image to a column vector, and last, store it in the matrix IMG. This IMG matrix has dimension m by n where m is the pixel of each image and n is the number of images. After obtaining the matrix, we can start the DMD algorithm.

First we let X takes the image data from IMG matrix except the last image, and X' be the matrix including image data from the second one to the last. Then we can do SVD on X1 and calculate the U,Σ, and V matrices. From the diagonal elements of the Σ matrix, we can see that there are only a small number of elements that matter in the entire matrix, so we take the small number as the rank r. And then, we can only work on the small dimension of the IMG matrix instead of the entire matrix.

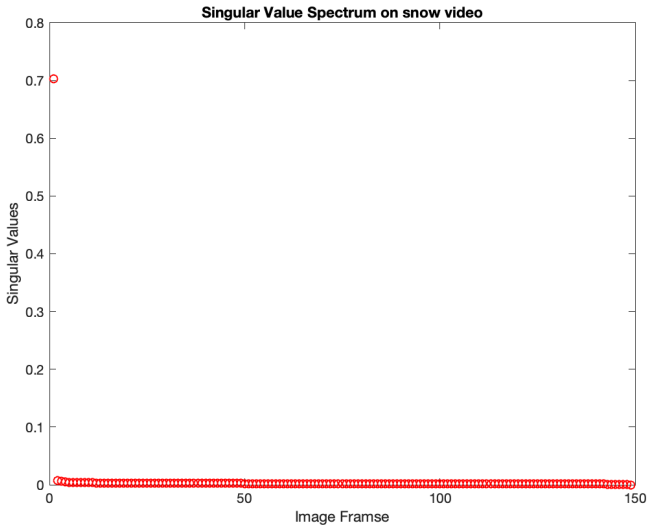
Now, we truncate the U,Σ, and V matrices with dimension r. Then I used the low rank matrices to reconstruct eigenvectors of A which is Φ. Then I calculated the initial b. After I have eigenvalues and eigenvectors for A, I am able to construct the solution using the formula in a for loop. And this matrix we got is the low-dimensional matrix with DMD modes.

In order to extract the foreground, we need to use the original IMG matrix to subtract the DMD matrix. However, there will be some negative elements in there. So I take out the negative elements, put them in a R matrix and subtract the R matrix from DMD matrix. Therefore, for the DMD matrix, I need to add back R matrix to make them satisfy:

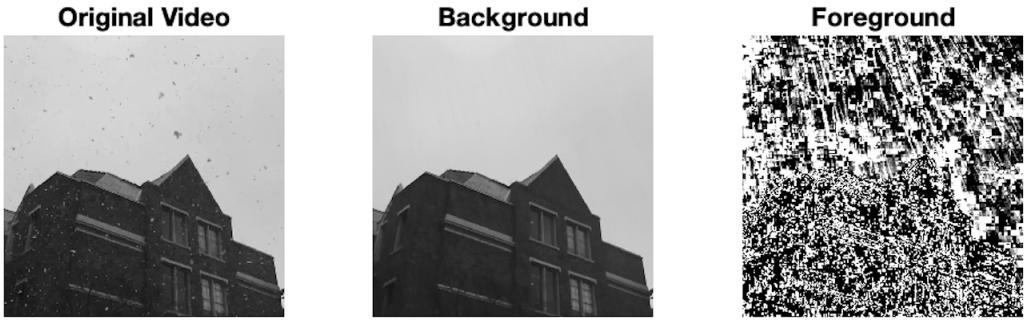
$$IMG = M_{DMD} + M_{sparse} \tag{21}$$

4 Computational Results

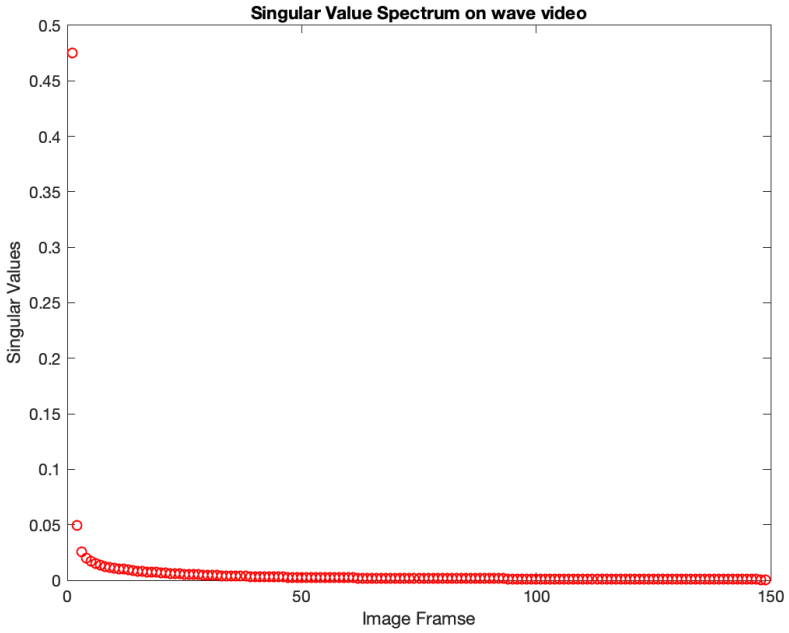
I tried two videos. The first one is a video of snow falling down. The background of this video is a building. For the singular value, I got the rank is 1 and $omega = 3.6214e-04$ which is pretty small.



And after running the DMD algorithm on this video, I separated the foreground and background of the video, and the result is:



As it is shown in the plots that background is pretty clear. This is because the video is steady without shaking. However, the foreground is kind of messy. I think it makes sense because snow flakes are small and snow is heavy on that day which means there are infinity many snow flakes. That is the reason why the foreground image is not clear enough and it has white dots everywhere. The white dots are the moving snow flakes.
Based on the discovery from the snow video, I tried the algorithm on another video which is the wave video. This time, the moving object is larger and more complete as a whole comparing to snow flakes. The singular value plot looks like this:



which also has rank 1 and $omega = 0.0204$. And the result plots are:

This time, the background is as clear as the snow video, and also, the foreground is more clear. Only the wave part is white and the background is blacked out.

In the theoretical part, it is said that if we subtract the R matrix from the sparse matrix, we are supposed to add it back to the low rank matrix. However, I tried that on this wave video. And the result looks like this:

As we can see in the picture that the wave is pretty clear in the background image and it can move when I view the video. This is caused by the sparse matrix R. Since the eigenvalues for both of the videos are pretty small, which means $M_dmd = M_lr_dmd$. This means that U,Σ, and V matrices contain redundancy of the system. If we truncate them properly, the redundancies will be removed. That is why we have $M_dmd = M_lr_dmd$ and we do not need to add the R_matrix back.
Based on the results of these two videos, we have successfully separated videos into backgrounds and foregrounds using DMD method.

5 Summary and Conclusions

From this assignment we can see that a high-dimensional data can be reduced to low-dimensional while still be well presented. This methods is very useful in analyzing huge dataset. By using DMD, we can do dimension reduction on a large scale dataset and work on a small portion of it. Take the two videos that I worked on in this assignment as an example. Originally, both of them have hundreds of images for each video; for each image, the dimension is almost 1 million. So the dimension of the entire video is n by m where n is 1 million and m is hundreds. After we did SVD on the matrix, we can discovery that only the first component matters, so we can truncate it down to dimension 1 and do DMD afterwards using truncated matrices, which makes the computation convenient and fast. Also, the entire video matrix can be reconstructed pretty well. Therefore, DMD is really useful in doing dimension reduction on high-dimensional data.

Appendix MATLAB codes

The following code is the one working on the snow video. It is the same for the wave video.

```
clear all;clc;

v = VideoReader('snow_cut.mp4');
nFrames = v.NumberOfFrames;
IMG = [];
for i=1:nFrames
img = rgb2gray(read(v,i)); % get one RGB image
img = reshape(img,720*720,1);
IMG = [IMG img];
end
IMG = double(IMG);
t2 = linspace(0,v.CurrentTime,nFrames+1);
t = t2(1:end-1);
dt=t(2)-t(1);
```

```
%%
X1 = IMG(:,1:end-1); X2 = IMG(:,2:end);

[U2,Sigma2,V2] = svd(X1, 'econ');
figure(1)
plot(diag(Sigma2)/sum(diag(Sigma2)),'ro')
ylabel('Singular Values')
xlabel('Image Framse')
title('Singular Value Spectrum on snow video')
print(gcf,'-dpng','rank_snow.png');
r=1;
U=U2(:,1:r); Sigma=Sigma2(1:r,1:r); V=V2(:,1:r);
Atilde = U'*X2*V/Sigma; % low rank subspace of A
[W,D] = eig(Atilde);
Phi=X2*V/Sigma*W; % DMD modes, project back out to real space

mu=diag(D);
omega=log(mu)/dt; % e^mu = omega, DMD eigenvalues

x1 = X1(:,1);
y0 = Phi\x1;

% reconstruct in time
modes = zeros(r,length(t));
for iter = 1:length(t)
    modes(:,iter) = (y0.*exp(omega*t(iter)));
end;
Xdmd = Phi * modes;

%%
X_sparse = IMG-Xdmd;
R_Matrix = X_sparse.*(X_sparse<0);
X_sparse_dmd = X_sparse - R_Matrix;
%%
figure(2)
for i = 1:nFrames
    subplot(1,3,1)
    img1 = uint8(IMG(:,i));
    imshow(reshape(img1,720,720))
    title('Original Video')
    subplot(1,3,2)
    img2 = uint8(Xdmd(:,i));
    imshow(reshape(img2,720,720))
    title('Background')
    subplot(1,3,3)
    img3 = real(X_sparse_dmd(:,i));
    imshow(reshape(img3,720,720))
    title('Foreground')
    drawnow
end
print(gcf,'-dpng','Images_snow.png');
```